110 案例实战: 千万级用户场景下的运营系统SQL调优 (2)

今天咱们继续来看这个千万级用户场景下的运营系统SQL调优案例,上次已经给大家说了一下业务背景以及SQL,这个SQL就是如下的一个:

SELECT COUNT(id) FROM users WHERE id IN (SELECT user_id FROM users_extent_info WHERE latest_login_time < xxxxx)

之前说了,系统运行的时候,肯定会先跑一下COUNT聚合函数来查查这个结果集有多少数据,然后再分批查询。结果就是这个COUNT聚合函数的SQL,在干万级大表的场景下,都要花几十秒才能跑出来,简直是大跌眼镜,这种性能,系统基本就没法跑了!

所以我们今天一起来分析一下这个SQL的执行计划,不过这里要给大家提醒一点的是,因为不同的 MySQL版本的执行计划可能都不一样,平时我们开发可能感觉不出来,但是实际上每个不同的MySQL版本都可能会调整生成执行计划的方式,所以同样的SQL在不同的MySQL版本下跑,可能执行计划都 不太一样。

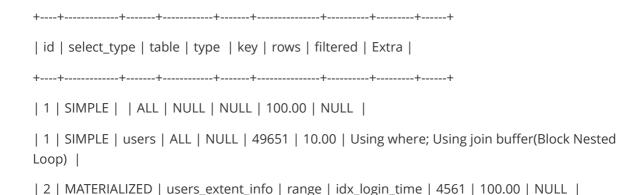
我们这里给出的执行计划是当时在我们的MySQL中得到的,可能大家自己拿同样的SQL去自己的 MySQL里没法还原出来这里的执行计划,但是没关系,大家重点学的是执行计划分析的思路,以及如何 从执行计划里看出性能问题所在,最后就是如何进行调优,重点是这个过程,没法还原出来执行计划,也是没关系的。

通过:

EXPLAIN SELECT COUNT(id) FROM users WHERE id IN (SELECT user_id FROM users_extent_info WHERE latest_login_time < xxxxx)

可以得到下面的执行计划,我们为了方便大家看,把执行计划简化了几个字段,保留了最关键的几个字段。

另外,给大家提示一点,下面的执行计划是当时我们为了调优,在测试环境的单表2万条数据场景下跑出来的执行计划,即使是5万条数据,当时这个SQL都跑了十多秒,所以足够复现当时的生产问题了,所以大家注意下执行计划里的数据量问题。



从上面的执行计划,我们可以清晰的看到这条SQL语句的一个执行过程

首先,针对子查询,是执行计划里的第三行实现的,他清晰的表明,针对users_extent_info,使用了idx_login_time这个索引,做了range类型的索引范围扫描,查出来了4561条数据,没有做其他的额外筛选,所以filtered是100%。

接着他这里的MATERIALIZED,表明了这里把子查询的4561条数据代表的结果集进行了物化,物化成了一个临时表,这个临时表物化,一定是会把4561条数据临时落到磁盘文件里去的,这个过程其实就挺慢的。

然后第二条执行计划表明,接着就是针对users表做了一个全表扫描,在全表扫描的时候扫出来了49651条数据,同时大家注意看Extra字段,显示了一个Using join buffer的信息,这个明确表示,此处居然在执行join操作???

接着看执行计划里的第一条,这里他是针对子查询产出的一个物化临时表,也就是,做了一个全表查询,把里面的数据都扫描了一遍,那么为什么要对这个临时表进行全表扫描呢?

原因就是在让users表的每一条数据,都要去跟物化临时表里的数据进行join,所以针对users表里的每一条数据,只能是去全表扫描一遍物化临时表,找找物化临时表里哪条数据是跟他匹配的,才能筛选出来一条结果。

第二条执行计划的全表扫描的结果表明是一共扫到了49651条数据,但是全表扫描的过程中,因为去跟物化临时表执行了一个join操作,而物化临时表里就4561条数据,所以最终第二条执行计划的filtered显示的是10%,也就是说,最终从users表里筛选出了也是4000多条数据。

这就是这条SQL语句的执行计划,不同MySQL版本可能是不一样的,甚至差别很大,所以大家没必要强求必须是要在自己本地可以还原出这个执行计划,但是大家重点是看明白我们这里对这个SQL语句的执行计划过程的一个分析。

End